NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

COST COMPARISON AMONG PROVABLE DATA POSSESSION SCHEMES

by

Stephen J. Bremer

March 2016

Thesis Advisor: Mark Gondree
Second Reader: Zachary Peterson

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	03-25-2016	Master's Thesis 06-11-2015 to 03-25-2016

4. TITLE AND SUBTITLE

COST COMPARISON AMONG PROVABLE DATA POSSESSION SCHEMES

5. FUNDING NUMBERS

6. AUTHOR(S)

Stephen J. Bremer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Provable data possession (PDP) provides mechanisms to efficiently audit the integrity of data held by third parties, like cloud service providers. While multiple PDP schemes have been proposed, there is no research to date that provides in-depth cost analysis for PDP. This research fills that gap by (1) collecting and analyzing cost data for four PDP schemes, (2) providing generic cost models (mathematical formulae expressing abstract models which can be used to infer future cost), and (3) comparing overall cost efficiency of each PDP scheme. For the schemes considered in this study, we find all have nearly identical costs in practice; however, sophisticated schemes designed with low communication complexity have higher preprocessing or storage costs which, depending on audit parameters, impact total scheme cost. We conclude that MAC-PDP and CPOR schemes are similar, whereas the cost of A-PDP becomes relatively expensive at large file sizes. Our basis cost projections show tagging, storing and auditing a file for one year at one audit per hour is at least $160 for a 1 GB file, $170 for a 1 TB file, and $2,000 for a 1 PB file using a cost model based on the Amazon S3 service.

14. SUBJECT TERMS

provable data possession, pdp, proof of retrievability, por, data integrity, data availability, cloud storage, cyber, cyber security, costs, cost comparison

15. NUMBER OF PAGES 59

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

COST COMPARISON AMONG PROVABLE DATA POSSESSION SCHEMES

Stephen J. Bremer
Lieutenant, United States Navy
B.A., Texas State University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: Mark Gondree
Thesis Advisor

Zachary Peterson
Second Reader

Cynthia Irvine
Chair, Cyber Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Provable data possession (PDP) provides mechanisms to efficiently audit the integrity of data held by third parties, like cloud service providers. While multiple PDP schemes have been proposed, there is no research to date that provides in-depth cost analysis for PDP. This research fills that gap by (1) collecting and analyzing cost data for four PDP schemes, (2) providing generic cost models (mathematical formulae expressing abstract models which can be used to infer future cost), and (3) comparing overall cost efficiency of each PDP scheme. For the schemes considered in this study, we find all have nearly identical costs in practice; however, sophisticated schemes designed with low communication complexity have higher preprocessing or storage costs which, depending on audit parameters, impact total scheme cost. We conclude that MAC-PDP and CPOR schemes are similar, whereas the cost of A-PDP becomes relatively expensive at large file sizes. Our basis cost projections show tagging, storing and auditing a file for one year at one audit per hour is at least $160 for a 1 GB file, $170 for a 1 TB file, and $2,000 for a 1 PB file using a cost model based on the Amazon S3 service.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AE	authenticated encryption
AWS	Amazon Web Services
CIO	Chief Information Officer
DISA	Defense Information Systems Agency
DOD	Department of Defense
NPS	Naval Postgraduate School
PDP	proof of data possession
POR	proof of retrievability
PRF	pseudo-random function
PRP	pseudo-random permutation
S3	Simple Storage Service
US	United States
USG	United States government
VM	virtual machine

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank my advisor, Dr. Mark Gondree, for his mentorship throughout the process of researching and writing my thesis. I am profoundly grateful for his patience and dedication. Any quality work to be found in my thesis is solely due to his careful guidance and advice.

I would also like to thank Dr. Pante Stanica whose sheer joy and enthusiasm for math sparked my own interest in the subject and provided me with the tools to understand the mathematics behind PDP.

I would especially like to thank my wife, Jodi, and our four children, Ethan, Matthew, Silas, and Eleanor. My wife does all things well, not least of which is her provision of graceful encouragement and support. I am undeserving of her devotion and self-sacrifice, but immensely thankful for it. Finally, my children are a constant source of delight and joy, which was a true blessing as I navigated this difficult task.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

The Department of Defense (DOD) has identified collaboration and improved access to information as key elements of future operational success. This need, coupled with the massive growth in data, has led the U.S. Navy and other DOD entities to invest in cloud storage capabilities in an effort to cope with "Big Data." In 2014, Terry Halvorsen, then-Navy Chief Information Officer (CIO), stated the Navy needs to move about half of its unclassified data into commercial cloud storage [1]. In late 2014, as acting DOD CIO, Halvorsen released a memo freeing DOD agencies to procure their own commercial cloud services, without using Defense Information Systems Agency (DISA), in an effort to speed up the migration process [2].

With the growing use of cloud storage solutions, there is a corresponding need for secure and efficient means of guaranteeing data integrity and availability.The Federal Cloud Computing Strategy of 2011 states that agencies should explicitly state security, availability, and quality requirements through service level agreements, and routinely monitor vendor compliance [3]. The DOD Cloud Computing Strategy of 2012 also establishes the requirement for cloud services to provide sufficient security to ensure the integrity and availability of DOD information [4]. In 2015, the DOD released its Cloud Computing Security Requirements Guide (SRG), which outlines the security requirements for DOD agencies procuring commercial cloud services. Among its recommendations are policies that would provide audit and accountability for data additions, deletions, and modifications [5]. Recent outages for well-known cloud storage providers, including Amazon S3 and Microsoft Azure, also underscore the need for a reliable and efficient auditing mechanism to ensure data availability and integrity as agencies migrate to the commercial cloud [6]–[8].

Proof of data possession schemes may provide the best mechanism to fulfill these demands to actively track vendor compliance and assure the integrity of data in storage. Through the use of cryptographic protocols, proof of data possession (PDP) schemes provide probabilistic guarantees that data on storage servers has not been maliciously or inadvertently deleted or altered. They claim to provide this guarantee at low cost to both the proving and verifying entities. Its guarantees are probabilistic and its asymptotic costs are strictly

sublinear in file size. This technology has not yet been implemented by or for a commercial service; however, there has been substantial research in PDP and other data integrity schemes over the past decade [9]–[32].

While multiple PDP schemes have been proposed, each with varying degrees of efficiency and security, there is no research to date that provides in-depth cost analysis comparing the real-world efficiencies of PDP schemes. All prior research has focused on two aspects of PDP schemes: providing high probability guarantees of data possession (security) while minimizing the size of the challenge and response (communication complexity). These are important criteria, especially in bandwidth-constrained environments; however, to date, no research has provided comparisons of PDP schemes in terms of real-world costs (time to generate proof, time to verify proof, time to tag, cost to store tag overhead, cost to run an audit service, cost to service requests from an audit service, etc).

Our research fills that gap by (1) collecting and analyzing cost data for four PDP schemes, (2) providing generic cost models (mathematical formulae expressing abstract models which can be used to infer future cost), and (3) comparing overall cost efficiency of each PDP scheme. Additionally, instead of measuring costs primarily in terms of the size of the query and response – a bandwidth concern – this research recognizes (a) the importance of processing time when evaluating the cost of a particular scheme, and (b) the asymmetric costs associated with some cloud cost models (e.g., PUTs are typically more expensive than GETs).

Based on our generic cost models, we show that the basis costs to audit are nearly identical for MAC-PDP, A-PDP, and CPOR, but tag and storage costs are different enough to have a significant impact on total cost among the schemes. We also show that the total cost of MAC-PDP and CPOR are similar, but A-PDP becomes expensive relative to the other schemes at large file sizes, due to its higher tag and storage costs. We show that the total basis cost (up-front cost to tag and cumulative cost storing and auditing) for one year at one audit per hour of a 1 GB file is under $1 for MAC-PDP, A-PDP, and CPOR, but that cost ranges from $4,400 to $38,700 across schemes for a 1 PB file.

CHAPTER 2:
Background

This research focuses on four specific PDP schemes: a simple MAC-based PDP scheme (MAC-PDP), the scheme described by Ateniese, Burns, Curtmola, Herring, Kissner, Peterson and Song (A-PDP) [33], the scheme decsribed by Ateniese, Pietro, Mancini and Tsudik (SEPDP) [34] and the scheme described by Shacham and Waters (CPOR) [35]. Our research is primarily concerned with building accurate cost models for each scheme based on experimental audit data. Below, we provide a generic description of PDP and a description of each PDP scheme considered in our experiments.

2.1 Proof of Data Possession

A PDP system can be divided into two generic phases: the set-up phase and the challenge phase. In the set-up phase, a client generates a public and private key pair, tags the file, and uploads the file and tag data to storage, deleting it from local storage. During the challenge phase, the client generates a challenge for a specified number of file blocks and sends the challenge to the prover. The prover uses the challenge to generate a proof of possession, which is returned to the client. The client then validates the proof, providing a probabilistic guarantee that the prover does or does not possess the client's file.

Following the notation of Juels and Kaliski [36] and Bower, Juels, and Oprea [37], a file M can be divided into n blocks, $M = \langle m_1, m_2, ..., m_n \rangle$. We let P denote the prover (server), V denote the verifier (client), η denote the file's identifier, and ω denote local client state. We represent unspecified values with a $\perp$ symbol. A generic PDP scheme can be considered a five-tuple of algorithms, $\langle \mathsf{KeyGen}, \mathsf{Tag}, \mathsf{Challenge}, \mathsf{Proof}, \mathsf{Verify} \rangle$, each described as the following.

$\mathsf{KeyGen}(1^k) \rightarrow (pk, sk)$. This algorithm is used by the client to generate random public and private keys by employing security parameter k.

$\mathsf{Tag}(M; pk, sk, \omega) \rightarrow M^*_\eta$. This algorithm is used by the client to process a file and produce verification tag data. It takes as input a public and private key pair (pk, sk) and file M. It generates a file ID η and returns M^*_η, the encoded file with verification tag

data. It also updates the client state ω to include and locally held data such as the file ID, file size, number of blocks, etc. The data M_η^* can be stored remotely.

Challenge$(\eta; pk, sk, \omega) \rightarrow c$. This algorithm is used by the client to produce a challenge c. This challenge will sent to the prover during an audit.

Proof$(\eta, M_\eta^*, c; pk) \rightarrow p$. This algorithm is used by the prover to demonstrate proof of possession of specified file blocks as a response to challenge c. It takes as input the remote, encoded data M_η^* and challenge c, to generate proof p.

Verify$(c, p, \eta; pk, sk, \omega) \rightarrow b \in \{0, 1\}$. This algorithm is used by the client to validate the proof p. It takes as input the public and private key pair (pk, sk), challenge c and proof p. Upon successful validation it returns 1, else it returns 0.

2.2 Constructions

In this section, we provide detailed descriptions of each PDP scheme employed in our study: MAC-PDP, A-PDP, SEPDP and CPOR.

2.2.1 MAC-PDP

The MAC-PDP scheme is defined below, following the description and notation from Shacham and Waters [35] and Riebel [38], adapted slightly for uniformity with the other schemes in Section 2.2.

Let f be a keyed pseudo-random function, as follows:

$$f : \{0, 1\}^* \times K_{prf} \rightarrow \mathbb{Z}_p$$

KeyGen$(1^k) \rightarrow (pk, sk)$. Choose a random secret key for a hash-based MAC function $k_{mac} \xleftarrow{R} K_{prf}$. The secret key is $sk = \langle k_{mac} \rangle$ and public key is $pk = \perp$.

Tag$(M; pk, sk, \omega) \rightarrow M_\eta^*$. The file is split into n blocks, $M = \langle m_1, m_2, ..., m_n \rangle$. Choose a random file ID η, where $\eta \in \mathbb{Z}_p$. For each block m_i, $(1 \leq i \leq n)$, generate tag $\sigma_i = MAC_{k_{mac}}(\eta || m_i)$. The data stored remotely is $M_\eta^* = \langle M, \{\sigma_i\}_{1 \leq i \leq n} \rangle$.

Challenge$(\eta; pk, sk, \omega) \rightarrow c$. Choose a random ℓ-element subset $I \subseteq [1, n]$ of indices. Let c be the set $\{i\}_{i \in I}$.

Proof$(\eta, M_\eta^*, c; pk) \rightarrow p$. For each $i \in c$, return to the verifier $p = \{(m_i, \sigma_i)\}_{i \in c}$.

Verify$(c,p,\eta;pk,sk,\omega) \rightarrow b \in \{0,1\}$. For each $i \in c$, check if $\sigma_i \stackrel{?}{=} MAC_{k_{mac}}(\eta||m_i)$. If all l checks are correct then return $b = 1$, else return $b = 0$.

2.2.2 A-PDP

The A-PDP scheme is defined below, following the description and notation from Ateniese *et al.* [33], adapted slightly for uniformity with the other schemes in Section 2.2.

Let H be a cryptographic hash function, h be a full-domain hash function, f be a pseudo-random function and π be a pseudo-random permutation (PRP) as follows (where κ, ℓ, λ are security parameters):

$$h : \{0,1\}^* \rightarrow QR_N \qquad (QR_N \text{ is the set of quadratic residues modulo } N)$$
$$f : \{0,1\}^\kappa \times \{0,1\}^{\log_2(n)} \rightarrow \{0,1\}^\ell$$
$$\pi : \{0,1\}^\kappa \times \{0,1\}^{\log_2(n)} \rightarrow \{0,1\}^{\log_2(n)}$$

KeyGen$(1^k) \rightarrow (pk, sk)$. Choose safe primes p, q, where $p = 2p' + 1$ and $q = 2q' + 1$. Let $N = pq$. Let g be a generator of QR_N, the set of quadratic residues modulo N. Let $v \stackrel{R}{\leftarrow} \{0,1\}^\kappa$. The public key $pk = \langle N, g \rangle$ and the secret key $sk = \langle e, d, v \rangle$, such that e is a large secret prime with $ed = 1 \pmod{p'q'}$, $e > \lambda$, $d > \lambda$.

Tag$(M; pk, sk, \omega) \rightarrow M^*_\eta$. The file is split into n blocks, $M = \langle m_1, m_2, ..., m_n \rangle$. For each block m_i, compute $T_{i,m_i} = (h(W_i) \cdot g^{m_i})^d \bmod N$, where $W_i = v||i$. The data stored remotely is $M^*_\eta = \langle M, \{(T_{i,m_i}, W_i)\}_{1 \leq i \leq n} \rangle$.

Challenge$(\eta; pk, sk, \omega) \rightarrow c$. To audit ℓ blocks of M, generate challenge $c = \langle \ell, k_1, k_2, g_s \rangle$, where k_1 and k_2 are random κ-bit keys, and $g_s = g^s \bmod N$ for random $s \stackrel{R}{\leftarrow} \mathbb{Z}^*_N$.

Proof$(\eta, M^*_\eta, c; pk) \rightarrow p$. For $1 \leq j \leq \ell$, generate indices $i_j = \pi_{k_1}(j)$ and coefficients $a_j = f_{k_2}(j)$. Compute $T = T^{a_1}_{i_1,m_{i_1}} \cdot \ldots \cdot T^{a_\ell}_{i_\ell,m_{i_\ell}} = (h(W_{i_1})^{a_1} \cdot \ldots \cdot h(W_{i_\ell})^{a_\ell} \cdot g^{a_1 m_{i_1} + \ldots + a_\ell m_{i_\ell}})^d \bmod N$. Compute $\rho = H(g_s^{a_1 m_{i_1} + \ldots + a_\ell m_{i_\ell}} \bmod N)$. The proof is $p = \langle T, \rho \rangle$.

Verify$(c, p, \eta; pk, sk, \omega) \rightarrow b \in \{0,1\}$. Let $\tau = T^e$. For $1 \leq j \leq \ell$, compute $i_j = \pi_{k_1}(j), W_{i_j} = v||i_j, a_j = f_{k_2}(j)$, and $\tau = \frac{\tau}{h(W_{i_j})^{a_j}} \bmod N$. If $H(\tau^s \bmod N) = \rho$ then return $b = 1$, else return $b = 0$.

2.2.3 CPOR

The CPOR scheme is defined below, following the description and notation from Shacham and Waters [35], adapted slightly for uniformity with the other schemes in Section 2.2.

Let f be a keyed pseudo-random function, as follows:

$$f : \{0,1\}^* \times \mathcal{K}_{prf} \rightarrow \mathbb{Z}_p$$

$\mathsf{KeyGen}(1^k) \rightarrow (pk, sk)$. Choose a random key $k_{enc} \xleftarrow{R} \mathcal{K}_{enc}$ for symmetric encryption scheme Enc, and a random HMAC key $k_{mac} \xleftarrow{R} \mathcal{K}_{mac}$. The secret key is $sk = \langle k_{enc}, k_{mac} \rangle$ and public key is $pk = \perp$.

$\mathsf{Tag}(M; pk, sk, \omega) \rightarrow M^*_\eta$. Given the file M, split M into n blocks, each s sectors long: $M = \langle m_{ij} \rangle_{\substack{1 \le i \le n \\ 1 \le j \le s}}$. Choose a PRF key $k_{prf} \xleftarrow{R} \mathcal{K}_{prf}$ and s random numbers $\alpha_1, ..., \alpha_s \xleftarrow{R} \mathbb{Z}_p$. Let $\tau_0 = \langle n || \mathsf{Enc}_{k_{enc}}(k_{prf} || \alpha_1 || \cdots || \alpha_s) \rangle$. The file tag is $\tau = \langle \tau_0 || \mathrm{MAC}_{k_{mac}}(\tau_0) \rangle$. For each $i, 1 \le i \le n$, compute

$$\sigma_i \leftarrow f_{k_{prf}}(i) + \sum_{j=1}^{s} \alpha_j m_{ij}$$

The data stored remotely is $M^*_\eta = \langle \{m_{ij}\}, \{\sigma_i\} \rangle$.

$\mathsf{Challenge}(\eta; pk, sk, \omega) \rightarrow c$. Choose a random ℓ-element subset $I \subseteq [1, n]$. For each $i \in I$ choose random $v_i \xleftarrow{R} \mathbb{Z}_p$. Let c be the set $\{(i, v_i)\}_{i \in I}$.

$\mathsf{Proof}(\eta, M^*_\eta, c; pk) \rightarrow p$. The prover parses c as $\{(i, v_i)\}$ and computes

$$\mu_j \leftarrow \sum_{(i,v_i) \in c} v_i m_{ij} \text{ for } 1 \le j \le s, \text{ and } \sigma \leftarrow \sum_{(i,v_i) \in c} v_i \sigma_i$$

The proof is $p = \langle \mu_k, \sigma \rangle_{1 \le k \le s}$.

$\mathsf{Verify}(c, p, \eta; pk, sk, \omega) \rightarrow b \in \{0,1\}$. Check $\sigma \stackrel{?}{=} \sum_{(i,v_i) \in c} v_i f_{k_{prf}}(i) + \sum_{j=1}^{s} \alpha_j \mu_j$. If equal then return $b = 1$, else return $b = 0$.

2.2.4 SEPDP

The SEPDP scheme is defined below, following the description and notation from Ateniese et al. [34], adapted slightly for uniformity with the other schemes in Section 2.2.

Let t be the number of possible challenges, H be a cryptographic hash function, AE be an authenticated encryption scheme, f be a keyed pseudo-random function and π be a keyed pseudo-random permutation, defined as follows:

$$H : \{0,1\}^* \rightarrow \{0,1\}^d$$
$$f : \{0,1\}^k \times \{0,1\}^{\log(t)} \rightarrow \{0,1\}^L$$
$$\pi : \{0,1\}^L \times \{0,1\}^{\log(n)} \rightarrow \{0,1\}^{\log(n)}$$

$\mathsf{KeyGen}(1^k) \rightarrow (pk, sk)$. Choose secret permutation key $W \xleftarrow{R} \{0,1\}^k$, master challenge nonce key $Z \xleftarrow{R} \{0,1\}^k$ and master encryption key $K \xleftarrow{R} \{0,1\}^k$. The secret key $sk = \langle W, Z, K \rangle$. The public key $pk = \perp$.

$\mathsf{Tag}(M; pk, sk, \omega) \rightarrow M^*_\eta$. Divide message M into n blocks. Choose the number t of possible random challenges and the number ℓ of block indices per verification. For each $1 \leq i \leq t$, generate the i-th tag as:

Generate a permutation key $k_i = f_W(i)$ and nonce $c_i = f_Z(i)$.

Compute the set of indices $\{i_j \in [1,n] \mid 1 \leq j \leq \ell\}$ where $i_j = \pi_{k_i}(j)$.

Compute token $v_i = H(c_i, m_{i_1}, \ldots, m_{i_\ell})$.

Encrypt the token $\sigma_i \leftarrow \mathsf{AE}_K(i, v_i)$.

The data stored remotely is $M^*_\eta = \langle M, \{i, \sigma_i\} \rangle$.

$\mathsf{Challenge}(\eta; pk, sk, \omega) \rightarrow c$. Generate the i-th challenge $c = \langle k_i, c_i \rangle$ by recomputing $k_i = f_W(i)$ and $c_i = f_Z(i)$.

$\mathsf{Proof}(\eta, M^*_\eta, c; pk) \rightarrow p$. Compute $z = H\left(c_i, m_{i_1}, \ldots, m_{i_\ell}\right)$ where $i_j = \pi_{k_i}(j)$. The proof is $p = \langle z, \sigma_i \rangle$.

$\mathsf{Verify}(c, p, \eta; pk, sk, \omega) \rightarrow b \in \{0,1\}$. Compute $v = \mathsf{AE}^{-1}_K(\sigma_i)$. If $v \stackrel{?}{=} (i, z)$ then return $b = 1$, else return $b = 0$.

2.3 Cost Complexity

The asymptotic communication complexity for each target PDP scheme is summarized in Table 2.1. While MAC-PDP affords a simple implementation, it is criticized for its relatively large communication complexity. Schemes like A-PDP, CPOR and SEPDP are designed with the goal of minimizing communication complexity [33]–[35].

Table 2.1: Asymptotic communication complexity of MAC-PDP, A-PDP, CPOR and SEPDP.

	Challenge	Proof
MAC-PDP	$O(\ell \log(n))$	$O(\ell(bs + k))$
A-PDP	$O(\log(\ell + 2\kappa + \log(N))$	$O(\log(N))$
CPOR	$O(\ell + (\log(n) + d))$	$O(\log(p))$
SEPDP	$O(L)$	$O(d + L)$

The block size bs is a function of file size and n, the number of file blocks.

2.4 Detection Probability

It is not the objective of this study to compare proofs associated with PDP schemes. To compare the cost of each scheme does require, however, selection of comparable parameters. There are at least three senses in which PDP schemes might be considered to be comparable.

Strength of Security. For a scheme, this is expressed as Pr[$forge$], the probability that a prover can get the verifier to accept a forged proof as valid (i.e., when it was computed without using some blocks involved in the challenge).

Strength of Audit. For a scheme, this is expressed as Pr[$audit$], the probability that a single audit will appear to succeed even when k of n blocks have been deleted. For many schemes, this is a combinatorial argument based on the probability that the ℓ random challenge indices are among the k blocks deleted.

Efficiency of Recovery. Some PDP schemes, often called proof of retrievability (POR) schemes, have the additional characteristic that the original file can be recovered even after some number of failed audits. For such a scheme, this is expressed as Pr[$recover$], the probability of retrieval after an ϵ fraction of audits have failed.

Comparison across schemes in these senses is problematic for a number of reasons: (i) schemes rely on different primitives (full-domain hash functions, authenticated encryption schemes, pseudorandom permutations) making parameter selection to achieve comparable Pr[$forge$] difficult; (ii) schemes have expressed these properties in slightly different adversarial models and employing slightly different arguments; (iii) arguments have been expressed in asymptotic terms rather than concrete terms, making parameter derivation dif-

ficult, especially when arguments employ bounds that are known to not be tight. Thus we do not select parameters in this study with the objective of providing absolute apples-to-apples comparison across schemes. Since the simple combinatorial arguments employed for Pr[$audit$] tend to be most reusable, we prioritize parameter selection for comparability in this sense. In some sense, this is a rather insignificant parameter since its probability can be driven arbitrarily low through repeated audits, due to exponential hardness amplification of passing a series of audits. At the same time, selection of this parameter may be most directly related to deriving policy on how often one performs audits. As we are interested in the recurring cost of audit, it is a natural parameter of our study to consider carefully. We leave open for future work parameter selection to facilitate fair comparison in terms of Pr[$forge$] and Pr[$recover$].

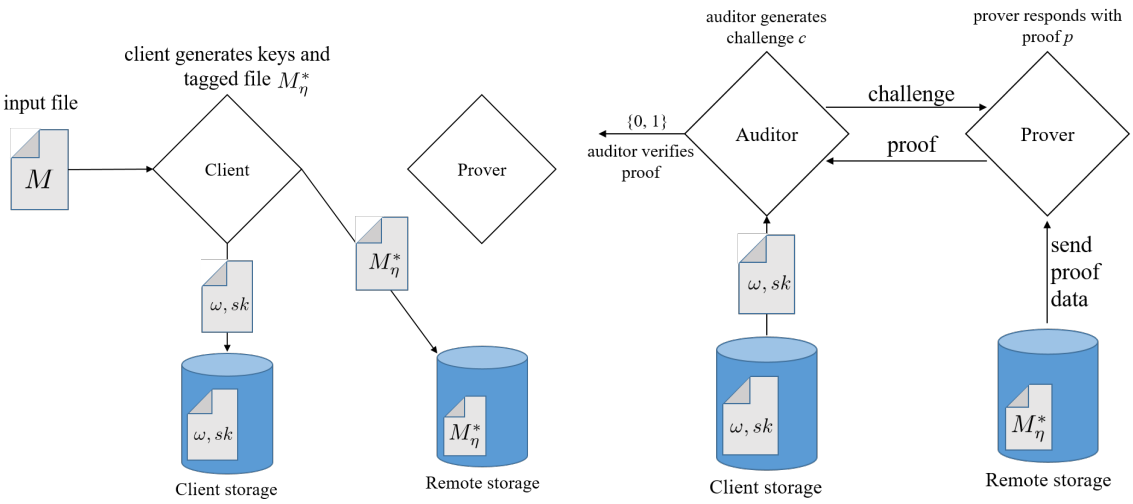
THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

This chapter discusses our experimental environment, methodology for how timing data is gathered, and implementation decisions for evaluating the performance of the PDP schemes under evaluation.

3.1 Experiment Environment

Our PDP experiments can be divided into two phases: a set-up phase and an audit phase. In the set-up phase, the client generates keys (pk, sk), generates a tagged file M_η^*, and sends M_η^* to remote storage (see Figure 3.1a). For the audit phase, the client generates a challenge c and sends it to the prover; the prover responds with a proof p, which is sent to the client; the client verifies the proof and indicates success or failure (see Figure 3.1b).



(a) Set-up phase of PDP protocol

(b) Audit phase of PDP protocol

Figure 3.1: Set-up and audit phases of PDP experiment.

Adapted from [33]: G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 598–609.

3.2 Measurements and Costs

It is important to define what system costs are measured in each of our experiments. We depict what operations are included in each of our measurements in Figure 3.2. Generally, we ignore costs associated with transfer time and service latency, focusing on significant, recurring computational costs.

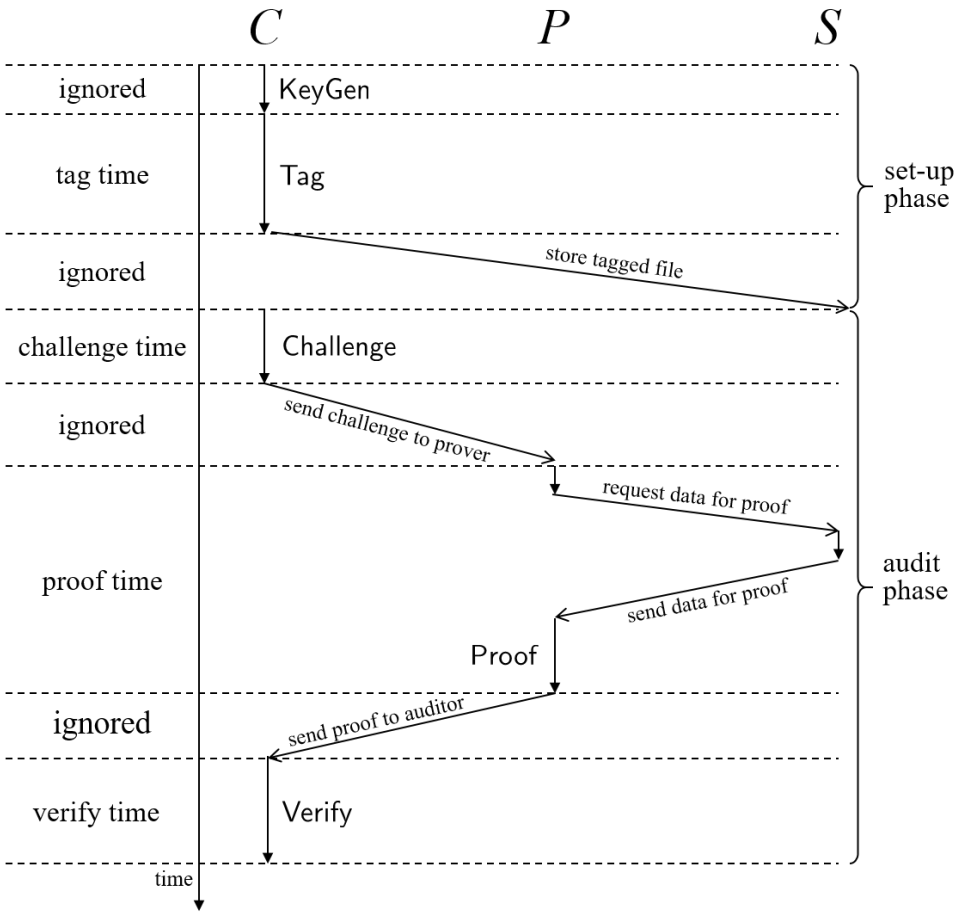


Figure 3.2: Timing measurement definitions, highlighting what operations and costs are included in each measurement.

In the set-up phase we do not measure the cost of generating keys (pk, sk). During tagging data, we ignore the cost of sending the file and tag data M_η^* to the storage server S. In the audit phase, we ignore the transfer time involved in sending the challenge to prover P and in returning the proof to client C. For proof generation, however, we include the time associated with retrieving challenge blocks from local or remote storage, including this as

part of the proof time. We believe the cost associated with parsing the challenge, retrieving the data required for the proof, and the cost of generating the proof itself are intimately related, and we combine these in our measurement.

3.3 Implementation

Our benchmark test is a single-threaded application written in C using the `libpdp` library [39], an open-source C library providing implementations for MAC-PDP, A-PDP, CPOR, and SEPDP. In all experiments, our benchmark application is run on Amazon Elastic Cloud (EC2). The client, auditor and prover are each run on the same EC2 instance: an c3.xlarge instance, running 64-bit Ubuntu Server 14.04 LTS using HVM virtualization. In other environments, these three parties might be separate hosts or owned by separate organizations (i.e., tagging and ingest performed by the data owner, and auditing performed by a third-party). As we have chosen to define tag, challenge and verify timing measurements, the properties of the network connecting these parties are irrelevant to our measurements and so we elect to run these parties on the same host. For each of our schemes, we conduct two types of benchmarks: using local data storage and using remote data storage. For local storage experiments, M_η^* is stored at the EC2 instance's local storage. For the remote storage experiments, M_η^* is stored to an Amazon S3 bucket.

Table 3.1: Default benchmark parameters used in our experiments.

MAC-PDP	$\ell = 460$, $k_{mac} = 20$ bytes
A-PDP	$\ell = 460$, $N = 1024$ bits, PRP $k_1 = 16$ bytes, PRF $k_2 = 20$ bytes
CPOR	$\ell = 460$, $k_{enc} = 32$ bytes, $k_{prf} = 20$, $k_{mac} = 20$ bytes, $\lambda = 80$, $p = 80$ bits, sector size = 9 bytes
SEPDP	$\ell = 460$, AE $K = 16$ bytes, PRP $W, Z = 16$ bytes, PRF $k_i = 20$ bytes, $t = 1$

Unless otherwise noted, $bs = 4096$ bytes and $fs = 2^{25}$ bytes.

Experiments are run sequentially, each time doubling block size or file size for a particular scheme. Pre-experiment trials in which the order of experiments are randomized demonstrated no discernible impact to our results; thus, we strongly believe our trials are independent and order of test execution had no impact to our results. Each experiment is performed using pre-generated, random input file data. Every experiment is repeated three

times (graphs in Chapter 4 show raw data from all three iterations). The default parameters used for each scheme is provided in Table 3.1.

CHAPTER 4:
Analysis

In this chapter, we analyze the timing data collected for each of the five major PDP algorithms: KeyGen, Tag, Challenge, Proof and Verify. Each algorithm is analyzed separately across all four schemes, including our expectations based on each algorithm, what the data actually show, and the cost model we have developed for each scheme and algorithm.

For each cost model, we employ the following notation:

bs, block size in bytes
fs, file size in bytes
ss, sector size in bytes
$c_0, c_1, \ldots$, model-specific constants.

For all the schemes, fs/bs yields the number of blocks in the file M. In each experiment, there is a point where the file size and block size are such that the total number of blocks falls below the default number of challenges selected for an audit. At this point, fewer computations are performed, resulting in faster algorithm times. Otherwise, all schemes approach some threshold where proof cost becomes constant. All model-specific constants are derived experimentally using least-squares approximation. Unless otherwise noted, all figure times are in seconds.

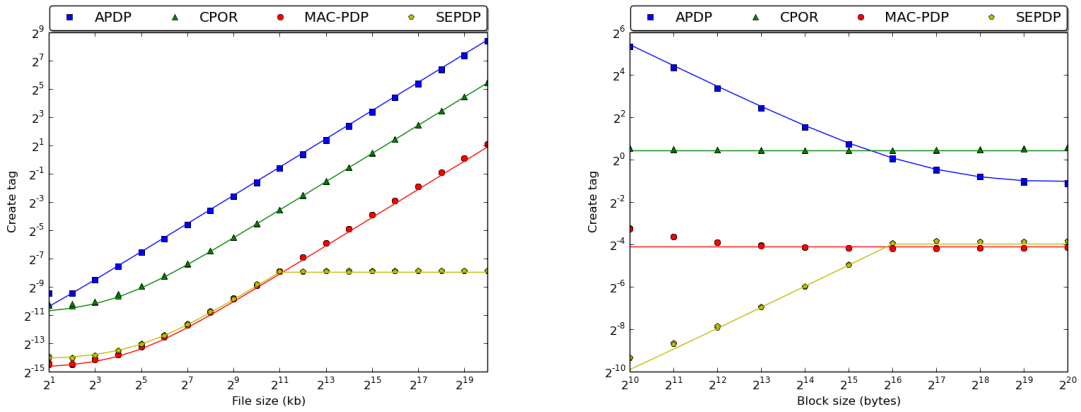
4.1 Tag File

In our experiments, there is no theoretical difference between running the Tag algorithm with local data or using AWS S3. Our measurements also bear this out.

4.1.1 MAC-PDP

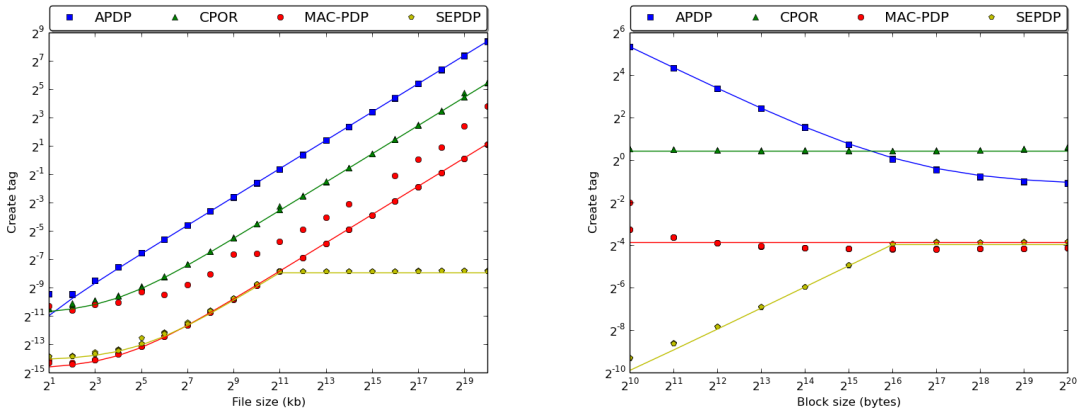
We observe that when block size is held constant and file size increases, the tag time increases linearly (see Figures 4.1a and 4.2a). When the file size remains constant and as the block size varies, the execution time is nearly constant (see Figures 4.1b and 4.2b).

This is explained in terms of MAC-PDP generating tags via a hash-based MAC on every



(a) File size vs. tag time (b) Block size vs. tag time

Figure 4.1: File and block size vs. tag time for local data experiments.



(a) File size vs. tag time (b) Block size vs. tag time

Figure 4.2: File and block size vs. tag time for S3 data experiments.

file block. Since the hash algorithm generates a digest through repeated operations on fixed-size blocks, the operation time should be proportional to the size of the input. We summarize these trends in Model 4.1, which expresses the tag time as proportional to the file size.

$$c_0 + c_1 \cdot fs \tag{4.1}$$

4.1.2 A-PDP

We observe that when block size is held constant and file size increases, the tag time increases linearly (see Figures 4.1a and 4.2a). When the file size is held constant and the block size increases, the tag time decreases linearly (see Figures 4.1b and 4.2b).

This is explained in terms of A-PDP generating tags through modular exponentiation on every block. As the file size grows, there will be more blocks to tag, resulting in increased execution time. As block size increases, there is a corresponding decrease in the number of blocks to tag. We summarize these trends in Model 4.2, which expresses the tag time as proportional to the file size and inversely proportional to the block size.

$$c_0 + c_1 \cdot fs/bs + c_2 \cdot bs + c_3 \cdot fs \tag{4.2}$$

4.1.3 CPOR

We observe that when block size is held constant and file size increases, the tag time increases linearly (see Figures 4.1a and 4.2a). When the file size is held constant and the block size increases, the tag time remains constant (see Figures 4.1b and 4.2b).

This is explained in terms of CPOR generating tags through nested loops of modular multiplication and addition. The number of loops is determined by the total number of sectors. An increase in file size results in a corresponding increase in the number of sectors. However, since changes in block size have little to no effect on the number of sectors, the algorithm times remain nearly constant as the block size varies. We summarize these trends in Model 4.3, which expresses the tag time as proportional to the file size and inversely proportional to the sector size.

$$c_0 + c_1 \cdot fs + c_2 \cdot fs/ss \tag{4.3}$$

4.1.4 SEPDP

We observe that when block size is held constant and file size increases, the tag time increases linearly up to a point, after which the tag time remains constant (see Figures 4.1a and 4.2a). When the file size is held constant and the block size increases, the tag time increases linearly up to a point, after which the tag time remains constant (see Figures 4.1b and 4.2b).

This is explained in terms of SEPDP generating tokens by calculating the hash of a specified number of blocks. The tag time, then, is proportional to the number of bytes being processed, which is determined by the number of blocks per token and the block size. The number of blocks per token is defined by the default security parameter ℓ, unless the block and file sizes are such that there are fewer blocks than the default parameter, in which case the token consists of all the blocks in the file. We summarize these trends in Model 4.4, which expresses the tag time as proportional to the total number of bytes processed per token.

$$(c_0 + c_1 \cdot \min((\min(fs/bs, \ell) \cdot bs, fs)) \cdot t \tag{4.4}$$

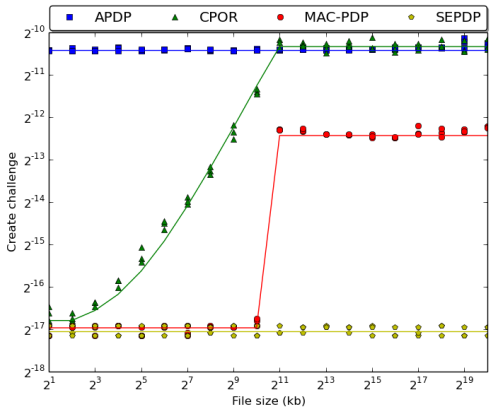
Above, $\min((\min(fs/bs, \ell) \cdot bs, fs)$ is essentially the number of bytes processed. When $fs/bs < r$, the entire file is processed to generate tokens.

4.2 Generate Challenge

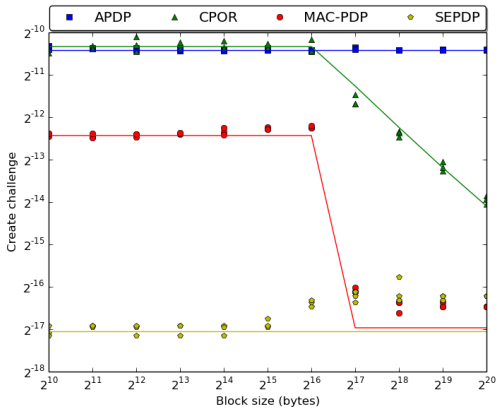
In our experiments, there is no theoretical difference between running the Challenge algorithm with local data or using AWS S3. Our measurements and resultant models also bear this out.

4.2.1 MAC-PDP

We observe that when block size is held constant and file size increases, the challenge time runs in constant time up to a point, after which it runs in a slower constant time (see Figures 4.3a and 4.4a). When the file size is held constant and the block size increases, the challenge time runs in constant time up to a point, after which it runs in a faster constant time (see Figures 4.3b and 4.4b).

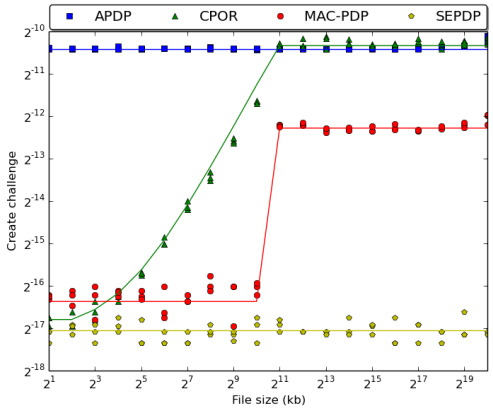


(a) File size vs. challenge time

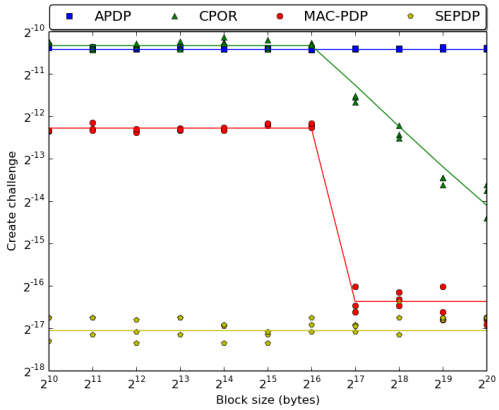


(b) Block size vs. challenge time

Figure 4.3: File and block size vs. generate challenge time for local data experiments.



(a) File size vs. challenge time



(b) Block size vs. challenge time

Figure 4.4: File and block size vs. generate challenge time for S3 data experiments.

This is explained in terms of ℓ and the total number of file blocks, given by fs/bs. When there are fewer total blocks than ℓ, then all indices are used for the challenge. However, when there are more blocks than ℓ, then the challenge indices must be chosen without replacement, which still runs in constant time, but takes longer than simply using all available indices. We summarize these trends in Model 4.5, which expresses the challenge time as one of two constants.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_0 \\ \lceil fs/bs \rceil \geq \ell &: c_1 \end{aligned} \tag{4.5}$$

4.2.2 A-PDP

We observe that generate challenge runs in constant time regardless of file or block size (see Figures 4.3 and 4.4). This is explained in terms of the A-PDP challenge being independent of the file or block size. We summarize these trends in Model 4.6, which expresses the challenge time as constant.

$$c_0 \tag{4.6}$$

4.2.3 CPOR

We observe that when block size is held constant and file size increases, the generate challenge time increases linearly up to a point, after which it runs in constant time (see Figures 4.3a and 4.4a). When the file size is held constant and the block size increases, the challenge time runs in constant time up to a point, after which it decreases linearly (see Figures 4.3b and 4.4b).

This is explained in terms of CPOR generating a random ℓ-element set for the challenge. As the file size increases, the size of this set increases, until the number of blocks exceeds ℓ. Similarly, when the block size increases to the point where there are fewer total blocks than ℓ, then the size of the challenge set will begin to decrease. We summarize these trends in Model 4.7, which expresses the challenge time as either constant or proportional to the total number of blocks.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_1 + c_2 \cdot fs/bs \\ \lceil fs/bs \rceil \geq \ell &: c_0 \end{aligned} \tag{4.7}$$

4.2.4 SEPDP

We observe that when block size is held constant and file size increases, generate challenge runs in constant time (see Figures 4.3a and 4.4a). As the file size is held constant and the block size increases, challenge runs in constant time up to a point, after which the run time is almost twice as slow (see Figures 4.3b and 4.4b).

The former trend is explained in terms of SEPDP recomputing k_i and c_i for the i-th challenge, neither of which is affected by the file size. We are unable to explain the latter trend. Nothing in the algorithm design suggests that block size should affect the run time, and we believe that the anomaly is an artifact of implementation, not a feature of the scheme. We summarize these trends in Cost Model 4.8, which expresses the challenge time as constant.

$$c_0 \tag{4.8}$$

4.3 Generate Proof

In our experiments, there is a noticeable difference between timing for the Proof algorithm using local data storage compared to using remote data storage using AWS S3. We analyze these two sets of experiments, separately.

For experiments interacting with S3, we observe that when block size is held constant and file size increases, the proof time increases linearly up to the point where the number of blocks exceeds ℓ, after which the proof time is constant (see Figure 4.7a). When the file size is held constant and the block size increases, the proof time is nearly constant up to the point where ℓ exceeds the number of blocks, after which the proof time decreases linearly (see Figure 4.7b).

This is explained in terms of each GET from S3 taking significantly more time than generating the proof itself (see Figure 4.6). Thus, the number of GETs dominates the trend. For MAC-PDP, A-PDP, and CPOR there is one GET for each challenged block and one GET for each corresponding tag (see Figure 4.5). This is summarized in Equation 4.9, which expresses the number of GETs as twice the total number of blocks or twice ℓ, whichever is less.

$$2 \cdot \min(fs/bs, \ell) \tag{4.9}$$

For SEPDP, there is one GET for each challenged block, but only one GET for the token corresponding to the i-th challenge (see Figure 4.5). This is summarized in Equation 4.10, which express the number of GETs as one more than the total number of blocks or one more than ℓ, whichever is less.

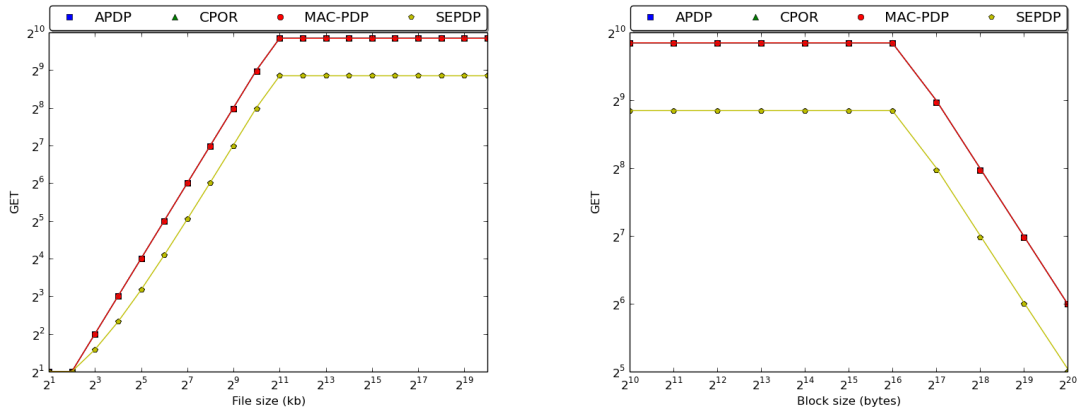
$$\min(fs/bs, \ell) + 1 \tag{4.10}$$

4.3.1 MAC-PDP

For local data experiments, we observe that when block size is held constant and file size increases, the proof time increases linearly up to the point where the number of blocks exceeds ℓ, after which the proof time is nearly constant, increasing slightly as the file size grows (see Figure 4.6a). When the file size is held constant and the block size increases, the proof time increases linearly up to the point where ℓ exceeds the number of blocks, after which the proof time is constant (see Figure 4.6b).

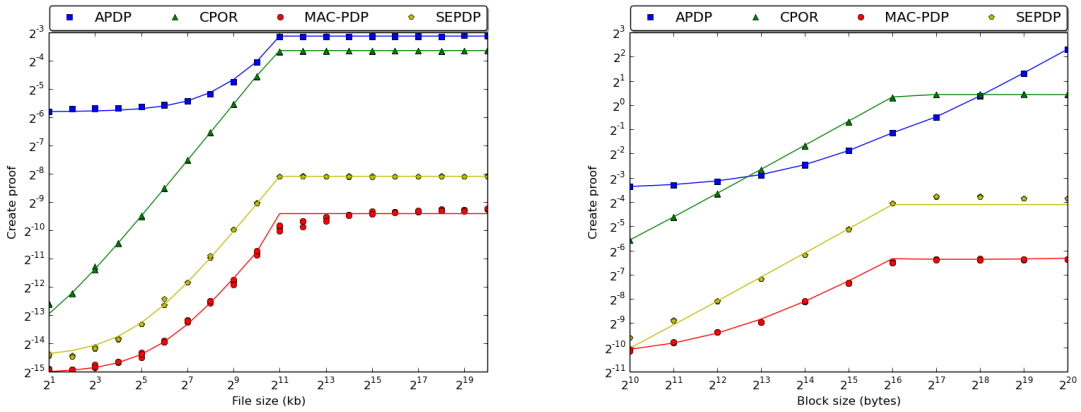
This is explained in terms of MAC-PDP generating a proof containing a message block and hash for each index in the challenge. The proof is dependent on the total number of bytes hashed. We summarize these trends in Model 4.11, which expresses the proof time as proportional to the total number of blocks, file size, and block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_0 + c_1 \cdot fs/bs + c_2 \cdot bs + c_3 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c_4 + c_5 \cdot bs \end{aligned} \tag{4.11}$$



(a) File size vs. GETs (b) Block size vs. GETs

Figure 4.5: File and block size vs. number of GETs from S3.



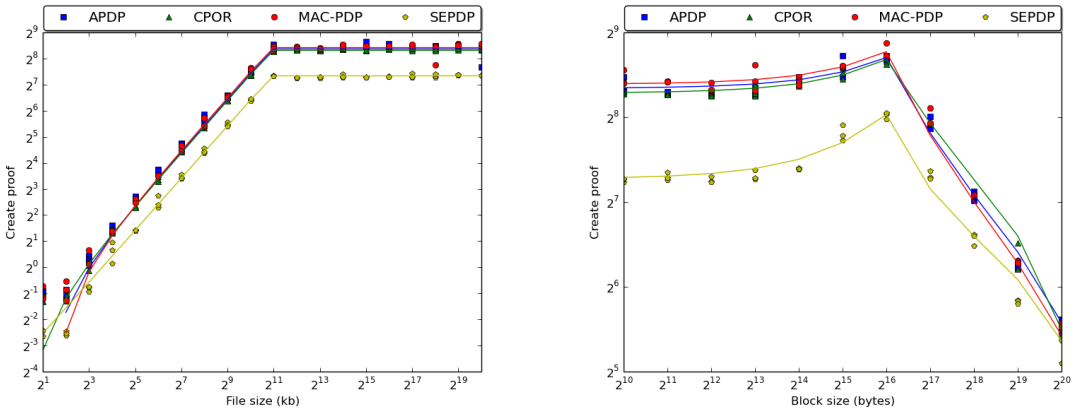
(a) File size vs. proof time (b) Block size vs. proof time

Figure 4.6: File and block size vs. generate proof time for local data experiments.

4.3.2 A-PDP

For local data experiments, we observe that when block size is held constant and file size increases, the proof time increases linearly up to the point where the number of blocks exceeds ℓ, after which the proof time remains constant (see Figure 4.6a). When the file size is held constant and the block size increases, the proof time increases linearly (see Figure 4.6b).

This is explained in terms of A-PDP generating proofs through modular exponentiation



(a) File size vs. proof time

(b) Block size vs. proof time

Figure 4.7: File and block size vs. generate proof time for S3 data experiments.

of ℓ message blocks. Thus the proof time will depend on the total number of challenge blocks as well as the size of each block. We summarize these trends in Model 4.12, which expresses the proof time as proportional to the number of blocks, file size, and block size or proportional to just the block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_2 + c_3 \cdot fs/bs + c_4 \cdot bs + c_5 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c_0 + c_1 \cdot bs \end{aligned} \tag{4.12}$$

4.3.3 CPOR

For local data experiments, we observe that when block size is held constant and file size increases, the proof time increases linearly up to the point where the number of blocks exceeds ℓ, after which the proof time remains constant (see Figure 4.6a). When the file size is held constant and the block size increases, the proof time increases linearly up the to the point where ℓ exceeds the number of blocks, after which the proof time remains constant (see Figure 4.6b).

This is explained in terms of CPOR generating the proof by computing μ_j and σ for each

of the indices in the challenge set. Additionally, μ_j includes modular multiplication of all the sectors of each challenge block. Therefore, the proof time increases with the indices in the challenge set, as well as when the block size increases. We summarize these trends in Model 4.13, which expresses the proof time as proportional to the number of blocks, file size, and block size, or proportional to just the block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c0 + c1 \cdot fs/bs + c2 \cdot bs + c3 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c4 + c5 \cdot bs \end{aligned} \tag{4.13}$$

4.3.4 SEPDP

For local data experiments, we observe that when block size is held constant and file size increases, the proof time increases linearly up to the point where the number of blocks exceeds ℓ, after which the proof time remains constant (see Figure 4.6a). When the file size is held constant and the block size increases, the proof time increases linearly up the to the point where ℓ exceeds the number of blocks, after which the proof time remains constant (see Figure 4.6b).

This is explained in terms of SEPDP generating the proof by computing the hash of all the message blocks for a particular token. The proof time is proportional, then, to the total number of bytes being hashed, given by the number of challenge blocks and block size. We summarize these trends in Model 4.14, which expresses the proof time as proportional to the total number of blocks, block size, and file size, or proportional to just the block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c0 + c1 \cdot fs/bs + c2 \cdot bs + c3 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c4 + c5 \cdot bs \end{aligned} \tag{4.14}$$

4.4 Verify Proof

In our experiments, there is no theoretical difference between running the Verify algorithm with local data or using AWS S3. Our measurements and resultant models also bear this out.

4.4.1 MAC-PDP

We observe that when block size is held constant and file size increases, the verify time increases linearly up to the point where the number of challenge blocks exceeds ℓ, after which it remains constant (see Figures 4.8a and 4.9a). When the file size is held constant and the block size increases, the verify time increases linearly up to the point where ℓ exceeds the total number of blocks, after which it remains constant (see Figures 4.8b and 4.9b).

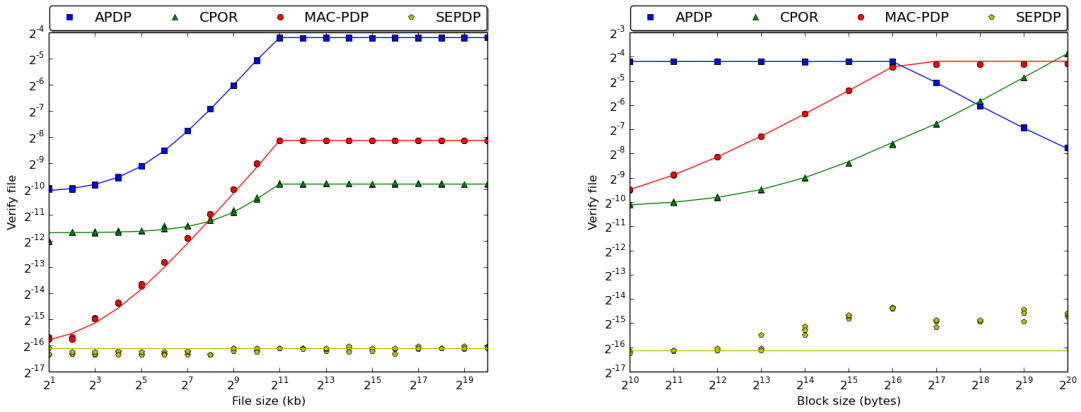
This is explained in terms of MAC-PDP verifying a proof by hashing each index in the challenge. Therefore, the verify time is dependent on the total number of bytes hashed. We summarize these trends in Model 4.15, which expresses the proof time as proportional to the file size or proportional to the block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_0 + c_1 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c_2 + c_3 \cdot bs \end{aligned} \tag{4.15}$$

4.4.2 A-PDP

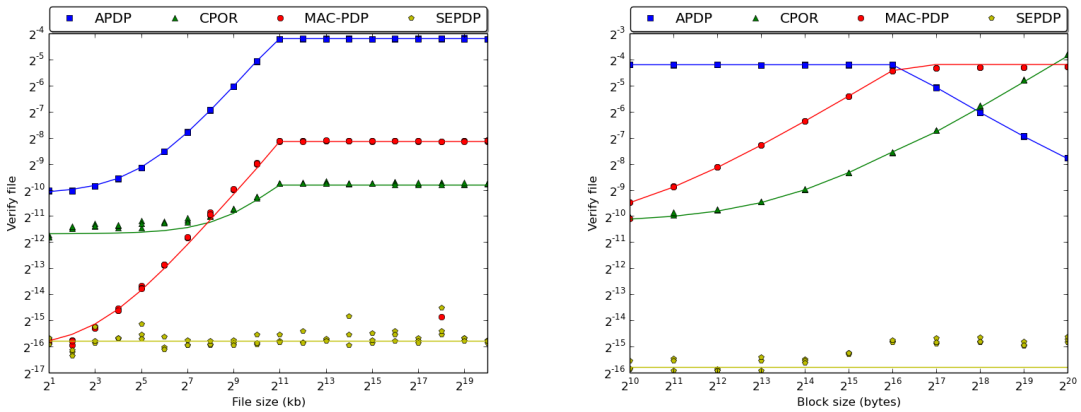
We observe that when block size is held constant and file size increases, the verify time increases linearly up to the point where the total number of blocks exceeds ℓ, after which it runs in constant time (see Figures 4.8a and 4.9a). When the file size is held constant and the block size increases, the verify time remains constant up to the point where ℓ exceeds the total number of blocks, after which it decreases linearly (see Figures 4.8b and 4.9b).

This is explained in terms of A-PDP verifying proofs by generating τ and comparing the hash of τ with ρ. Since τ is computed by generating ℓ hashes, the algorithm time will be proportional to the total number of blocks that were challenged. We summarize these



(a) File size vs. verify time (b) Block size vs. verify time

Figure 4.8: File and block size vs. verify proof time for local data experiments.



(a) File size vs. verify time (b) Block size vs. verify time

Figure 4.9: File and block size vs. verify proof time for S3 data experiments.

trends in Model 4.16, which expresses the verify time as constant or proportional to the total number of blocks.

$$\begin{aligned} \lceil fs/bs \rceil &< \ell : c_0 \\ \lceil fs/bs \rceil &\geq \ell : c_1 + c_2 \cdot fs/bs \end{aligned} \tag{4.16}$$

4.4.3 CPOR

We observe that when block size is held constant and file size increases, the verify time increases linearly up to the point where the total number of blocks exceeds ℓ, after which it runs in constant time (see Figures 4.8a and 4.9a). When the file size is held constant and the block size increases, the verify time increases linearly (see Figures 4.8b and 4.9b).

This is explained in terms of CPOR verifying the proof by summing $\alpha_j \mu_j$ for all sectors of each block being challenged. As the file size grows, the number of sectors for each challenge increases. As the block size grows, the number of sectors per block increases. We summarize these trends in Model 4.17, which expresses the verify time as proportional to the number of blocks, file size, and block size, or proportional to just the block size.

$$\begin{aligned} \lceil fs/bs \rceil < \ell &: c_0 + c_1 \cdot fs/bs + c_2 \cdot bs + c_3 \cdot fs \\ \lceil fs/bs \rceil \geq \ell &: c_4 + c_5 \cdot bs \end{aligned} \tag{4.17}$$

4.4.4 SEPDP

We observe that when block size is held constant and file size increases, the verify time remains constant (see Figures 4.8a and 4.9a). When the file size is held constant and the block size increases, the verify time remains constant up to a point, after which the verify time runs about twice as slow (see Figures 4.8b and 4.9b).

This is explained in terms of the SEPDP verify algorithm decrypting σ_i and comparing it with the proof. The decryption time should not be dependent on file size. Additionally, the decryption time should not be dependent on block size, and we believe that the anomaly is an artifact of implementation, not a feature of the scheme. We summarize these trends in Model 4.18, which expresses the verify time as a constant.

$$c_0 \tag{4.18}$$

4.5 Total Cost

We break costs down into three basic categories for analysis: (1) the cost to tag, which includes the computational costs to compute the tag and the PUT costs of uploading the tag; (2) the cost to store the tag; (3) the audit cost, which includes the computational cost to challenge, prove, and verify, and the GET costs associated with retrieving file blocks and tags during those operations.

SEPDP is not depicted on the cost graphs because its use of audit tokens does not compare well with the other schemes. Whereas MAC-PDP, APDP, and CPOR all support an unlimited number of audits once the file is tagged, the number of audits for SEPDP is chosen in advance. Thus a total cost graph for SEPDP will depend on the desired frequency of audits before a file needs to be retagged.

We note that the costs in our results should be thought of as *minimal* costs. We have ignored auditor costs associated with waiting for a response from the prover, as well as wake-up costs for the prover when it receives a proof request, which we do not measure as part of our experiments (see Figure 3.2). Measuring these costs would reflect network latency and implementation-specific details we do not believe to be strongly related to PDP. Also, in a scaled implementation of PDP, where multiple audits are performed for clients, simultaneously, the downtime costs may not be consequential. Thus the basis costs we depict do not reflect actual costs, but can accurately reflect cost comparisons among the schemes.

We chose to implement our benchmark tests on Amazon Web Services (AWS); however, there are several alternatives with comparable pricing schemes and storage options. For example, Microsoft Azure Blob storage, Google Cloud Storage, and Rackspace Cloud Files all have similar storage services and pricing schemes as Amazon. The AWS S3 storage pricing scheme is shown in Table 4.1[1].

[1]Prices were obtained from https://aws.amazon.com/s3/pricing as of March 2016.

Table 4.1: Amazon Web Services S3 standard storage pricing scheme.

	Cost / GB
First 1 TB / month	$0.0300
Next 49 TB / month	$0.0295
Next 450 TB / month	$0.0290
Next 500 TB / month	$0.0285
Next 4000 TB / month	$0.0280
Over 5000 TB / month	$0.0275

Table 4.2: Comparison of cloud providers remote storage limitations.

	Max object size	Max PUT size	Max metadata size
Amazon S3	5 TB	5 GB	2 KB
Microsoft Azure	195 GB	64 MB	8 KB
Google Cloud Storage	5 TB	5 TB	unspecified
Rackspace	5 GB	5 GB	4 KB

4.5.1 Tag Costs

Tag costs consist of the cost to generate the tag and the PUT costs associated with uploading the file to storage (see Figure 4.10). These costs resemble the trends we observed for computational costs associated with generating a tag (see Figure 4.1a), with A-PDP being the most expensive, followed by CPOR, and MAC-PDP. The approximate basis costs to tag a file range from a fraction of a cent to $3 for a 1 GB file; $0.13 to $20 for a 1 TB file; and $135 to $20,400 for a 1 PB file.

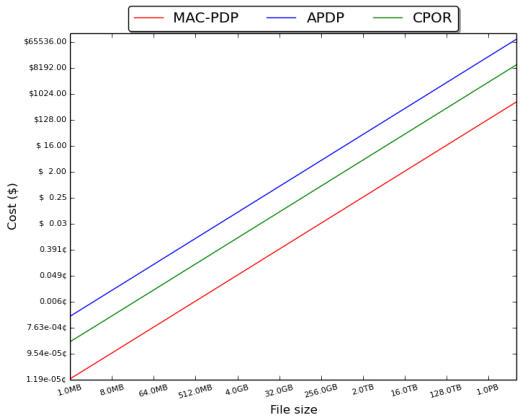
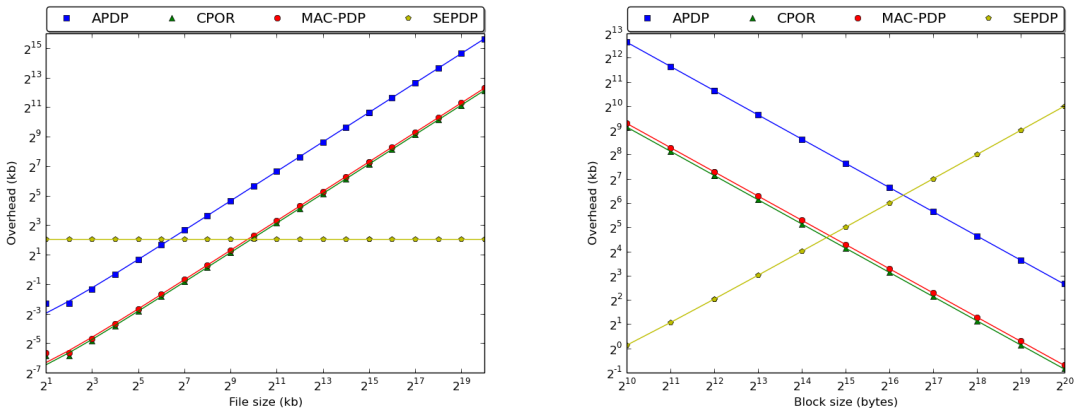


Figure 4.10: Cost to tag, based on tag algorithms and AWS EC2 pricing

4.5.2 Storage Costs

We calculate the storage cost for each scheme (see Figure 4.12) based on their corresponding tag sizes (see Table 4.3). As the file size increases the tag file overhead increases linearly for MAC-PDP, A-PDP, and CPOR, but remains constant for SEPDP; however, as the block size increases, the tag file overhead decreases linearly for MAC-PDP, A-PDP, and CPOR, but increases linearly for SEPDP (see Figure 4.11). Since A-PDP has the largest tag size, it has the highest storage cost. MAC-PDP and CPOR have almost the same tag size and, therefore, very similar storage costs.



(a) File size vs. overhead

(b) Block size vs. overhead

Figure 4.11: File and block size vs. tag file overhead.

Table 4.3: Tag file overhead and tag size for each scheme (bs = 4096 bytes).

	Total tag file overhead (% fs)	Tag size (bytes)
A-PDP	4.864%	204
MAC-PDP	0.477%	20
CPOR	0.429%	18

We investigated the option of storing tags as metadata to reduce cost; however, all the storage providers we reviewed included metadata as part of the overall file size. Additionally, at the time of publication, AWS S3 limits metadata storage to 2KB. The maximum file sizes at which the tags can be stored as metadata on AWS S3 are shown in Table 4.4.

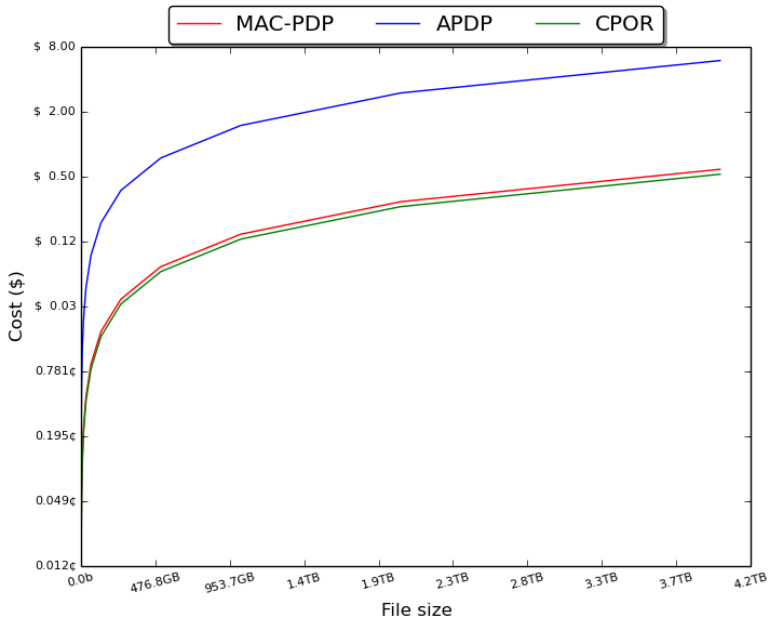


Figure 4.12: Cost to store tag, based on scheme tag overhead and AWS S3 pricing

Table 4.4: Maximum file sizes at which tags can be stored as metadata on AWS S3.

	File size
MAC-PDP	428 kb
A-PDP	41 kb
CPOR	476 kb
SEPDP	0 kb

4.5.3 Audit Costs

We calculate the total audit cost (see Figure 4.13) by determining the number of GETs and computational cost to generate a challenge, generate a proof, and verify the proof. Since the proof time is significantly larger than the challenge or verify times (compare Figure 4.7 with Figures 4.4 and 4.9), we are not surprised to find the proof time dictates the audit cost trends. Additionally, the differences in proof times observable in the local data experiments (see Figure 4.6) nearly disappear in the S3 experiments due to the relatively larger times required to communicate with S3 and transfer proof data. As a consequence of the communication time common to all schemes, the audit costs are nearly identical for

MAC-PDP, A-PDP, and CPOR.

It is worth noting that the audit cost for SEPDP is approximately half that of the three other schemes. The SEPDP proof scheme has fewer GETs than the other schemes since it only retrieves a single tag file in each audit, instead of a tag per challenge block, as in the other schemes.

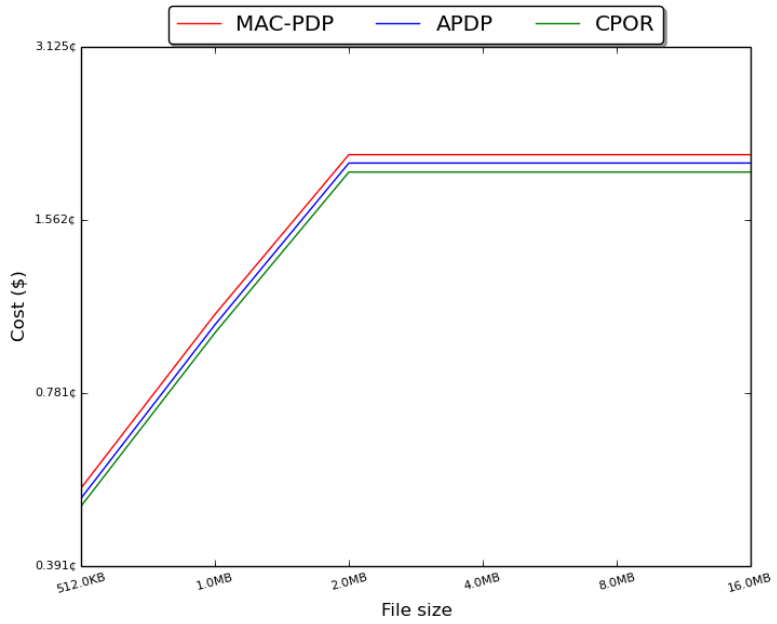


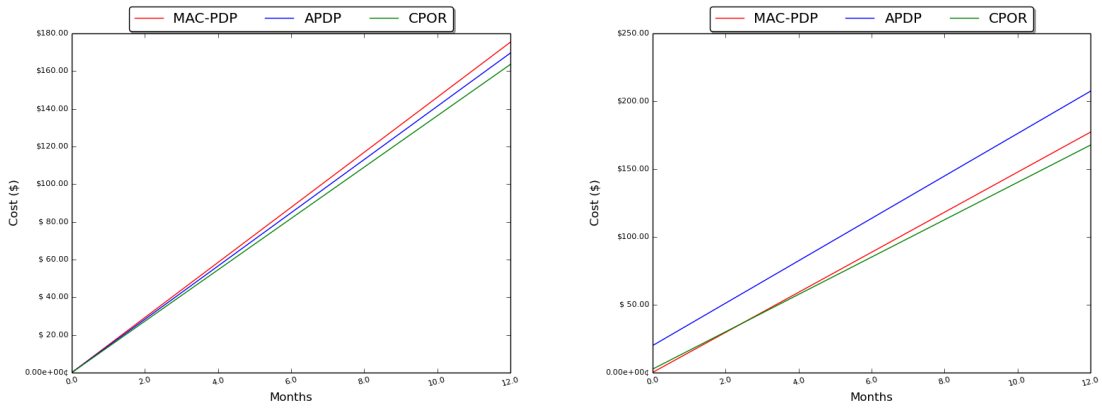
Figure 4.13: Cost to audit, based on audit cost models and AWS EC2 and S3 pricing

4.5.4 Combined Cost Scenarios

We observe that the monthly cost to store and audit once per hour is nearly identical for all schemes until the storage costs begin to dominate at larger file sizes, after which A-PDP becomes much more expensive than MAC-PDP and CPOR (see Figure 4.15).

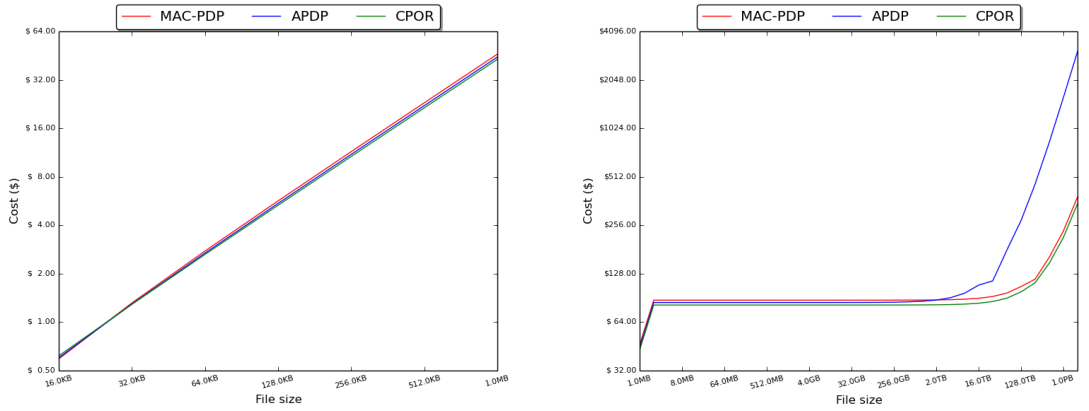
Since the audit costs are nearly identical for all three schemes, the tag and storage costs have the most significant impact on the total cost of each scheme. Figures 4.14a and 4.14b show the up-front cost to tag and cumulative cost storing and auditing a 1 GB and 1 TB file, respectively, at one audit per hour each month. For the 1 GB file, the tag and storage costs are less significant and the slightly higher audit cost of MAC-PDP can be observed

at one year of audits; however, the high tag and storage costs of the 1 TB file dominate, resulting in a higher cost for the A-PDP scheme. The following are approximate basis costs incorporating up-front cost to tag and cumulative cost storing and auditing at one audit per hour for one year: $160 to $175 for a 1 GB file; $170 to $230 for a 1 TB file; and $2,000 to $38,700 for a 1 PB file.



(a) Tag, storage, and audit costs for 1 GB file (b) Tag, storage, and audit costs for 1 TB file

Figure 4.14: Cumulative tag, storage, and audit costs for one audit per hour.



(a) File size vs. storage and audit costs (b) File size vs. storage and audit costs

Figure 4.15: File size vs. storage and audit costs for files at one audit per hour for one month.

CHAPTER 5:
Conclusion

We have developed generic cost models for four PDP schemes, which can be used to infer future cost. Additionally, we have shown that audit costs of some sophisticated PDP schemes (A-PDP, CPOR) are nearly identical to those of the simple MAC-PDP scheme; whereas, tag and storage costs have a significant impact on total cost differences among the schemes. We conclude that the total cost of MAC-PDP and CPOR are comparable, whereas the cost of A-PDP becomes expensive relative to the other schemes at large file sizes. Our preliminary experimentation shows audit cost for SEPDP is about half the other schemes; however, the scheme is limited to a finite number of audits.

From cost projections based on generic models for MAC-PDP, A-PDP, and CPOR, we find the basis cost for tagging is less than $1 for a 1 GB file; $0.13 to $20 for a 1 TB file; and $135 to $20,400 for a 1 PB file. The monthly basis cost for storage is a fraction of a cent for a 1 GB file; $0.13 to $1.50 for a 1 TB file; and $130 to $1,500 for a 1 PB file. The cost for a single audit is approximately $0.02 for files larger than 2 MB. Combined cost projections incorporating up-front cost to tag and cumulative cost storing and auditing at one audit per hour for one year show basis costs of $160 to $175 for a 1 GB file; $170 to $230 for a 1 TB file; and $2,000 to $38,700 for a 1 PB file.

5.1 Future Work

While our benchmark tests covered a limited number and type of PDP implementations, future studies could compare schemes that incorporate erasure codes, dynamic data, or distributed file system storage, among other variants. Our experiments ignored costs associated with transfer time and service latency, focusing instead on computational costs. Follow-on work could separate the client, auditor, and prover in order to measure the communication costs between each entity. Lastly, follow-on work could compare costs choosing different security parameters. In our experiments, we selected security parameters designed to normalize comparison in terms of the strength of audit (as defined in Chapter 2). Future work could select parameters to facilitate scheme comparison in terms of other properties, such as strength of security and efficiency of recovery.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] H. Kenyon, "Navy eyes cloud storage," *InformationWeek*, 18 March 2014. Available: http://www.informationweek.com/government/cloud-computing/navy-eyes-cloud-storage/d/d-id/1127748 [Last accessed: 6 March 2016].

[2] S. Lyngaas, "Halvorsen formalizes new dod cloud procurement policy," 2014. [Online]. Available: https://fcw.com/articles/2014/12/17/dod-cloud-policy.aspx

[3] V. Kundra, "Federal cloud computing strategy," 2011. [Online]. Available: http://www.dhs.gov/sites/default/files/publications/digital-strategy/federal-cloud-computing-strategy.pdf

[4] T. Takai, "Dod cloud computing strategy," 2012. [Online]. Available: http://dodcio.defense.gov/Portals/0/Documents/DoD%20Cloud%20Computing%20Strategy%20Final%20with%20Memo%20-%20July%205%202012.pdf

[5] Defense Information Systems Agency, "Department of defense cloud computing security requirements guide," 2015. [Online]. Available: http://iase.disa.mil/cloud_security/Documents/u-cloud_computing_srg_v1r1_final.pdf

[6] B. Butler. What broke Amazon's cloud. *Network World*, 23 September 2015. Available: http://www.networkworld.com/article/2985554/cloud-computing/what-brought-down-amazon-s-cloud.html [Last accessed: 6 March 2016].

[7] J. Jackson, "Human error root cause of November Microsoft Azure outage," *ComputerWorld*, 17 December 2014. Available: http://www.computerworld.com/article/2860833/human-error-root-cause-of-november-microsoft-azure-outage.html [Last accessed: 6 March 2016].

[8] T. Jones, "Rackspace tackles bug with full xen reboot," *TechTarget*, 30 September 2014. Available: http://searchcloudcomputing.techtarget.com/news/2240231810/Rackspace-tackles-bug-with-full-Xen-reboot [Last accessed: 6 March 2016].

[9] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 319–333.

[10] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 1–34, June 2011.

[11] D. Cash, A. Kupcu, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 279–295.

[12] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, 2010, pp. 31–42.

[13] B. Chen and R. Curtmola, "Robust dynamic provable data possession," in *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2012, pp. 515–525.

[14] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 411–420.

[15] Y. Deswarte, J.-J. Quisquater, and A. Saidane, "Remote integrity checking," in *IFIP TC11/WG11.5 Sixth Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, S. Jajodia and L. Strous, Eds., 2004, pp. 1–11.

[16] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2009, pp. 213–222.

[17] M. Etemad and A. Kupcu, "Transparent, distributed, and replicated dynamic provable data possession," in *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 1–18.

[18] C. Hanser and D. Slamanig, "Efficient simultaneous privately and publicly verifiable robust provable data possession from elliptic curves," in *Security and Cryptography (SECRYPT), 2013 International Conference on*, July 2013, pp. 1–12.

[19] R. S. Kumar and A. Saxena, "Data integrity proofs in cloud storage," in *Communication Systems and Networks (COMSNETS), Third International Conference on*, Jan 2011, pp. 1–4.

[20] J. Li, M. Krohn, D. Mazieeres, and D. Shasha, "Secure untrusted data repository (sundr)," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004, pp. 9–9.

[21] J. Li, X. Tan, X. Chen, D. S. Wong, and F. Xhafa, "Opor: Enabling proof of retrievability in cloud computing with resource-constrained devices," *Cloud Computing, IEEE Transactions on*, vol. 3, no. 2, pp. 195–205, 2015.

[22] F. Liu, D. Gu, and H. Lu, "An improved dynamic provable data possession model," in *Cloud Computing and Intelligence Systems (CCIS), IEEE International Conference on*, Sept 2011, pp. 290–295.

[23] H. Liu, P. Zhang, and J. Liu, "Public data integrity verification for secure cloud storage," *Journal of Networks*, vol. 8, no. 2, 2013. [Online]. Available: http://ojs.academypublisher.com/index.php/jnw/article/view/jnw0802373380

[24] Z. Mo, Y. Zhou, and S. Chen, "A dynamic proof of retrievability (por) scheme with big-oh (logn) complexity," in *Communications (ICC), IEEE International Conference on*. IEEE, 2012, pp. 912–916.

[25] T. S. J. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 12–33. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2006.80

[26] Y. Wang, Q. Wu, D. S. Wong, B. Qin, S. S. M. Chow, Z. Liu, and X. Tan, "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *19th European Symposium on Research in Computer Security (ESORICS)*, M. Kutylowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 326–343. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11203-9_19

[27] B. Wang, B. Li, and H. Li, "Panda: Public auditing for shared data with efficient user revocation in the cloud," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 92–106, Jan 2015.

[28] J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1717–1726, Aug 2015.

[29] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proceedings of the 2013 international workshop on Security in cloud computing*. ACM, 2013, pp. 19–26.

[30] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Communications and Network Security (CNS), IEEE Conference on*, Oct 2013, pp. 145–153.

[31] Y. Zhang and M. Blanton, "Efficient dynamic provable possession of remote data via balanced update trees," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. New York, NY, USA: ACM, 2013, pp. 183–194.

[32] Q. Zheng and S. Xu, "Fair and dynamic proofs of retrievability," in *Proceedings of the First ACM Conference on Data and Application Security and Privacy*. ACM, 2011, pp. 237–248.

[33] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 598–609.

[34] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, 2008, p. 9.

[35] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Advances in Cryptology—ASIACRYPT 2008*. Heidelberg: Springer, 2008, pp. 90–107.

[36] A. Juels and B. S. Kaliski, Jr., "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2007, pp. 584–597.

[37] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. New York, NY, USA: ACM, 2009, pp. 43–54.

[38] K. C. Riebel-Charity, "Developing a library for proofs of data possession in charm," master's thesis, Naval Postgraduate School, Monterey, 2013.

[39] M. Gondree and Z. Peterson. libpdp, a library for proofs of data possession. [Online]. Available: https://github.com/gondree/libpdp

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California